

Solvers for $\mathcal{O}(N)$ Electronic Structure in the Strong Scaling Limit

Nicolas Bock* and Matt Challacombe†

Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87544

Laxmikant V. Kalè

Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign

(Dated: March 31, 2014)

We present two parallel implementations within the OpenMP and Charm++ frameworks of the important spectral projection $\mathcal{O}(N)$ quantum chemical solver, with our recently introduced Sparse Approximate Matrix Multiply (SpAMM) as kernel. We find that the error in the energy under a finite SpAMM tolerance is well controlled while at the same time reducing the computational complexity of the solver to $\mathcal{O}(n \lg n)$. We present parallel scaling studies of water cluster systems on up to 24,576 CPU cores. We find that the standard load balancing strategies of Charm++ lead to impressive fine grained parallelism at this scale, which is approaching 1,000 CPU cores per water molecule for the smaller systems.

Keywords: Sparse Approximate Matrix Multiply; Sparse Linear Algebra; SpAMM; Reduced Complexity Algorithm; Quantum Chemistry; N-Body; Matrices with Decay

I. INTRODUCTION

The electronic structure problem involves the solution of an integro-differential equation, and exhibits a computational complexity of $\mathcal{O}(N^3)$ or higher. In systems that exhibit a band gap, one can exploit the fact however, that the matrix elements of the density matrix decay exponentially with spatial separation, opening up the possibility of thresholding the matrix elements, and to convert the density matrix into a sparse matrix. Recently we proposed a novel approach to $\mathcal{O}(N)$ linear algebra, the Sparse Approximate Matrix Multiply (SpAMM) [1, 2]. SpAMM is based on the N-Body solvers framework, and achieves a complexity reduction through thresholding of the product space as opposed to the vector space of the matrix.

N-Body methods are an emerging class of solvers across a broad range of applications. The N-Body framework is based on the general idea of trees (binary and higher dimensional) and an appropriate range-query function which allows to drop contributions dynamically, reducing the computational complexity of the solver often to near linear. Examples include database join operation [3–6], gravitational force summation [7–9], the Coulomb interaction [10, 11], the fast Gauß transform [12–16], adaptive mesh refinement [17–19], the fast multipole method [20–23] and visualization applications [24–30].

Recently, we introduced a generalized N-Body approach to linear algebra [1], and demonstrated a high-performance implementation in serial [2], extending the already broad list of applications for N-Body solvers to approximate linear algebra.

In this contribution we show how the N-Body frame-

work can lead naturally to an over decomposition of the problem into largely independent chunks. The fine-grained parallelism is exploited in a hybrid OpenMP/MPI approach, in which the MPI part is managed by the Charm++ run-time [31]. It is shown how a capable load balancing run-time leads to good parallel scaling up to a large number of processors.

$\mathcal{O}(N)$ electronic structure at scale is typically only shown in the weak scaling limit up to a few molecules per CPU core. This is due to the fact that matrix blocking based on atom blocks is used and that it is challenging to parallelize typical sparse matrix technology. Buluç *et al.* [32–34] has made nice progress in this direction, however, due to the inherent limitations of the CSR algorithm, they have to randomize the matrix element order, and hence lose any chance to exploit any atom ordering.

SpAMM relies on a complexity reduction in product space. The product space is 3-dimensional and offers a much bigger opportunity of task over decomposition compared to regular 2-dimensional vector space thresholding. Operating in a higher dimensional space allows us to push electronic structure towards the strong scaling limit.

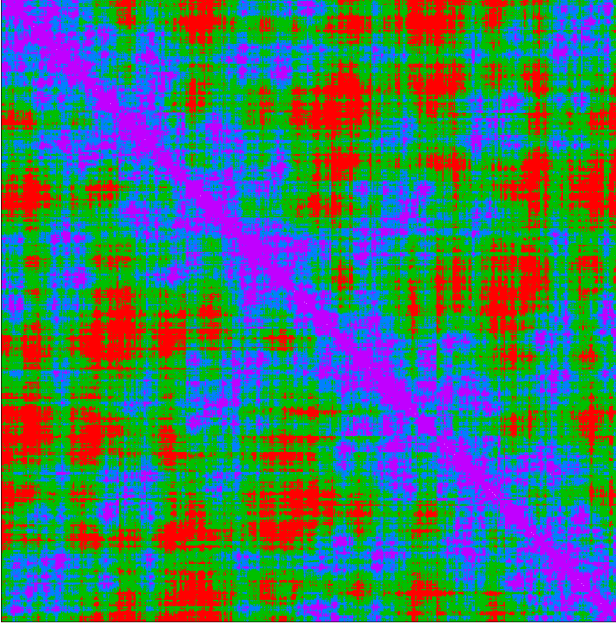
II. SPARSE APPROXIMATE MATRIX MULTIPLY

The SpAMM algorithm in its naïve recursive form was introduced by us previously [1] and we will give here only a short summary of the key ideas, and refer to the original article for more details: SpAMM falls in the class of generalized N-Body solvers and approaches matrix-matrix multiplication via a tree representation of the matrix and the recursive construction of the product in 3-dimensional convolution space under a tolerance. To this end, matrices are recursively divided into quadrants by bisection of the row and column spaces up to a predefined basic block size N_b . This procedure leads to a quadtree

*Electronic address: nbock@lanl.gov

†Electronic address: mchalla@lanl.gov

FIG. 1: The decay of matrix element magnitudes of a converged spectral projector (density matrix) for a $(\text{H}_2\text{O})_{300}$ water cluster at the RHF/6-31G** level of theory ($n = 7500$), where the molecular geometry has been reordered with a space filling Hilbert curve. The different colors indicate different matrix element magnitudes; red: $[0, 10^{-8})$; green: $[10^{-8}, 10^{-6})$; blue: $[10^{-6}, 10^{-2})$; violet: $[10^{-2}, 1]$, corresponding to approximate exponential decay.



representation of the matrix in which tree nodes contain links to their children nodes and leaf nodes store the matrix elements of a full $N_b \times N_b$ basic matrix. Each node is decorated with the Frobenius norm,

$$\|A\|_F = \left(\sum_{ij} |A_{ij}|^2 \right)^{1/2}, \quad (1)$$

of its submatrix. If a particular submatrix is zero the tree terminates at this point. Multiplying two such matrices proceeds in a recursive fashion under the condition that the product of the matrix norms be above a chosen tolerance τ ,

$$C_{ij} \leftarrow \begin{cases} A_{ik} B_{kj} & \text{iff } \|A_{ik}\| \|B_{kj}\| > \tau \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

For matrices with decay such as the quantum chemical density matrix shown in Fig. 1, this procedure leads to a complexity reduction of the product from $\mathcal{O}(N^3)$ to $\mathcal{O}(N)$ [1]. Note that the atomic coordinates of the underlying physical system of Fig. 1 were ordered along a space filling curve (we chose the Hilbert curve) to achieve the clustering of small matrix elements. Note also that it is the decay coupled with clustering of small matrix elements which leads to a complexity reduction.

We conclude this section by noting that the complexity reduction of the matrix product comes from thresholding in the 3-dimensional product space through Eq. 2 and not from thresholding in the 2-dimensional matrix element space.

A. Scalable Parallelism through Task Over-Decomposition

SpAMM operates recursively in the 3-dimensional product space of the matrix product which offers significant opportunities for parallelization through over-decomposition, demonstrated in Alg. 1 using the OpenMP programming model: Before recursing on line 6, an untied OpenMP task is created, deferring further execution of this recursion step until the task is scheduled to execute in an available thread. Potential data dependencies are either expressed through OpenMP `taskwait` statements or explicit locks. This procedure leads to the creation of many light weight tree tasks and compute intensive leaf node tasks, which are load balanced by the OpenMP run-time.

Algorithm 1 The OpenMP SpAMM algorithm, recursively multiplying matrices $C \leftarrow A \times B$ under a SpAMM tolerance τ . The function matrix arguments are pointers to tree nodes.

```

1: function MULTIPLY( $\tau$ , tier,  $A$ ,  $B$ ,  $C$ )
2:   if tier < depth then
3:     for all  $\{i, j, k \mid C_{ij} \leftarrow A_{ik} B_{kj}\}$  do
4:       if  $\|A_{ik}\| \|B_{kj}\| > \tau$  then ▷ Eq. 2
5:         Create untied OpenMP task
6:         MULTIPLY( $\tau$ , tier+1,  $A_{ik}$ ,  $B_{kj}$ ,  $C_{ij}$ )
7:       end if
8:     end for
9:     OpenMP taskwait
10:  else
11:    Acquire OpenMP lock on  $C$ 
12:     $C \leftarrow C + A \times B$  ▷ Dense product, e.g. BLAS
13:    Release OpenMP lock on  $C$ 
14:  end if
15: end function

```

Unfortunately, the OpenMP programming model is currently restricted to shared memory architectures.

B. Charm++ Programming model

Charm++ [31] is a mature and proven run-time environment that provides perhaps the closest equivalent to the OpenMP task based parallelism discussed in the previous section on a distributed memory platform. Charm++ has consistently demonstrated high performance at scale for a wide range of applications [35–43], making it a very attractive option.

The concept of a task is mapped onto a Charm++ “chare”, a C++ object which encapsulates migratable

code and data. In addition to so called “singleton” chares, it is possible to define multi-dimensional arrays of chares, whose elements can be load balanced transparently by the run-time. As the load balancers implemented in Charm++ work only with chare arrays, we can not adopt the simple recursive task creation scheme used in Alg. 1 and instead recast the recursion as an iteration over chare arrays, see Alg. 2.

Algorithm 2 The SpAMM algorithm in the Charm++ programming model. Instead of recursion, the algorithm iterates over chare arrays.

```

function MULTIPLY( $\tau$ ,  $A$ ,  $B$ ,  $C$ )
  for  $t \geq 0 \wedge t < d$  do
    end for
end function

```

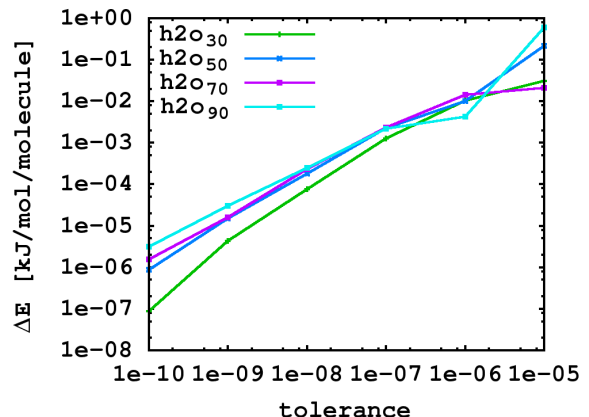
Describe this a bit better. Maybe algorithm for charm approach? Explain why the node code is using OpenMP.

We implemented the matrix quadtree as a set of 2-dimensional chare arrays of size $2^t \times 2^t$, where t is the tier. Just as in the OpenMP implementation Alg. 1, each chare stores the Frobenius norm of its sub-matrix. Since the chare arrays are full¹, it is not necessary to store children pointers, since the array indices of the children nodes can be reconstructed. The matrix product is implemented as a set of 3-dimensional chare arrays. The multiply implementation iterates over the matrix tiers, and tests the matrix norm products in each multiply element through a broadcast message to chare array elements. In case a product falls below the SpAMM tolerance, the multiply elements underneath are flagged as disabled and short circuited in the next tier. This trick can not eliminate the $\mathcal{O}(N^2)$ computational cost in each multiplication tier, but it reduces the prefactor significantly albeit with a dense storage complexity. Finally, at the leaf tier, the multiply elements perform matrix products and store the result in a temporary location, eliminating any data dependence between results. In the last step, the temporary matrices are summed into the final matrix C .

During chare migration, the data in a chare is packed and unpacked (serialized). Allocating the OpenMP multiply in a standard linked list of non-contiguous memory chunks risks degrading performance through inefficient memory copy operations. Instead, a sufficiently sized memory chunk is allocated upon creation and the linked list structure inside stores address offsets instead of pointers, making the chunk position independent (another requirement for chare migration since the actual address space is not guaranteed to be identical after a migration).

¹ In the current Charm++ implementation (v6.6.0-rc2), communication aware load balancers required full chare arrays, see Issue #445, <https://charm.cs.illinois.edu/redmine/issues/445>.

FIG. 2: The absolute error of the energy after roughly 40 iterations of the spectral projection method for different water clusters in RHF/6-31G**.



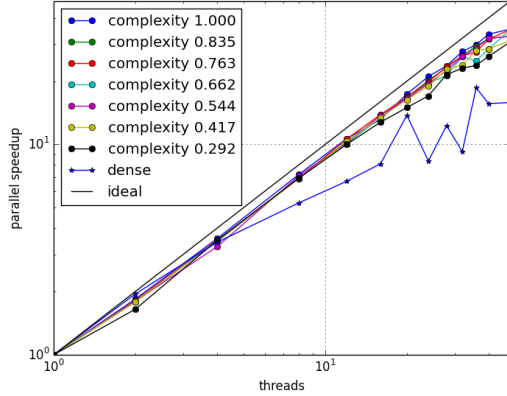
III. METHODS

In a typical quantum chemistry calculation, the solution of an integro-differential equation is sought by expanding the electronic wavefunctions of a molecule into a set of atomic basis functions yielding a matrix eigenvalue problem [44], or spectral projection [45–50]. The exact form and number of functions in the basis set and hence the number of expansion coefficients per atom depend on the atom type and the basis set. Because SFC ordering is applied atom-wise, matrix elements are clustered by magnitude, grouped in submatrices given by the atom type and basis set. In the following the 6-31G** basis was chosen for quantum chemical matrices corresponding to Hilbert curve ordered water clusters. The basis set introduces submatrices of different sizes (6-31G**: H 5×5 , O 15×15), representing the native granularities embedded in the matrix structures. The matrices were generated at the Restricted Hartree-Fock (RHF) level of theory with **FreeON**, a suite of programs for $\mathcal{O}(N)$ quantum chemistry [51] and are fully converged and without sparsification ($\epsilon = 0$), *i.e.* the matrices are fully dense but exhibit the approximately exponential decay pattern shown in Fig. 1. The sequence of water clusters corresponds to standard temperature and pressure and have been used in a number of previous studies [10, 52–58].

Previously, we reported SpAMM errors as the max norm of the difference between the SpAMM product and the dense reference product [2]. As the test systems are identical, per product errors are as well, and we focus here on the application of SpAMM in the second order spectral projection (SP2) quantum chemical solver [59] and report the energy difference between exact SP2 and SpAMM SP2, $\Delta E = |E_{\tau=0} - E_{\tau}|$, shown in Fig. 2.

The OpenMP scaling studies were run on a 48 core, 4-socket AMD Opteron 6168 (Magny Cours) system running at 1.9 GHz. The Charm++ scaling studies were run on the mustang cluster at LANL, which consists of 1,600

FIG. 3: Parallel scaling of the OpenMP implementation, shown in Alg. 1 on a 48 core AMD Opteron 6168 (Magny Cours) system. For comparison, the scaling of a dense DGEMM operation using the threaded MKL library is shown labeled “dense”. The OpenMP run-time is clearly balancing the SpAMM tasks efficiently and achieves $> 90\%$ parallel efficiency at 24 threads and $> 70\%$ parallel efficiency at 48 cores.



dual socket AMD Opteron 6167 (Magny Cours) nodes with 24 cores per node running at 2.3 GHz, and a total of 38,400 cores.

IV. RESULTS

A. OpenMP scaling

In a first series of tests, we applied the OpenMP SpAMM implementation of Alg. 1 to the density matrix of the $(\text{H}_2\text{O})_{110}$ water cluster using different SpAMM tolerance values. Shown in Fig. 3 is the parallel efficiency, *i.e.* the ratio of serial time vs. time on P threads. For comparison, the parallel scaling of the dense DGEMM implementation of the MKL library is shown as well.

We find that the parallel efficiency of SpAMM is very good for all SpAMM tolerance values tested. At 24 threads, SpAMM exhibits almost 90% efficiency which drops to a little over 70% at 48 cores. The dense MKL implementation on the other hand can not take advantage of the CPU cores to the same extent and its efficiency drops to as little as 50% at 48 cores. The significant drop in parallel performance at high thread counts of the MKL is presumable precipitated by memory bandwidth restrictions.

B. Charm++ Scaling

In a second series of tests, we used the water clusters we described previously [2]. As we have previously reported the error per matrix multiply, we report here the error in the energy after a fully converged spectral projection.

FIG. 4: The wall time of the first iteration of the SP2 algorithm at scale at $\tau = 10^{-10}$.

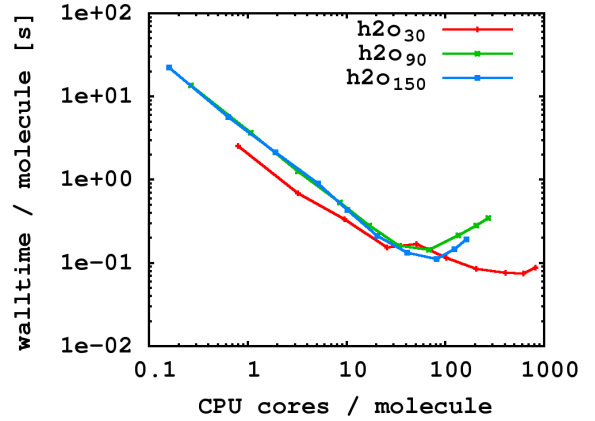
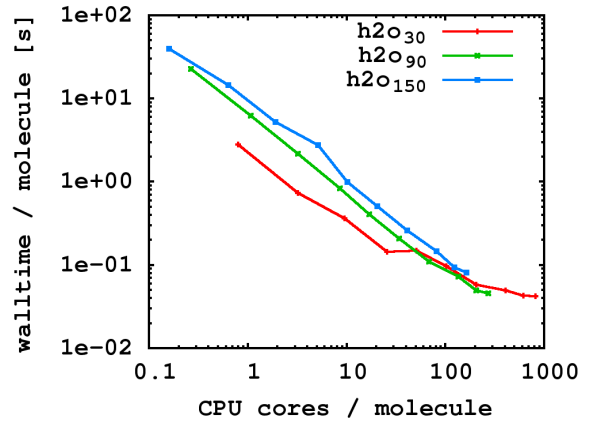


FIG. 5: The wall time of the last iteration of the SP2 algorithm at scale at $\tau = 10^{-10}$.



This takes about 40 iterations. Fig. 2 shows the energy error as a function of SpAMM tolerance. As expected, an increasing tolerance leads to an increasing error in the energy. However, the error is well controlled by the tolerance. For practical applications, a tolerance of $\tau \approx 10^{-8}$ seems reasonable.

We performed the spectral projection calculation with different tolerances on an increasing number of nodes of the mustang computer cluster at LANL. The initial data distribution for the first iteration is given by the Charm++ default load balancer. After each iteration of the SP2 algorithm, Charm++ rebalances the work and data. To demonstrate the efficiency of the load balancer, we show walltime vs. cores for a tight threshold of 10^{10} in Fig. 4. While parallel scaling is good for few processors, the initial data distribution leads to poor parallel performance at larger core counts. After a few iterations however, the load balancer dynamically migrates the matrix and multiply element chores in such a way as to improve performance all the way out to 24,576 cores (1024 nodes on mustang), Fig. 5.

FIG. 6: The wall time of the first iteration of the SP2 algorithm at scale at $\tau = 10^{-8}$.

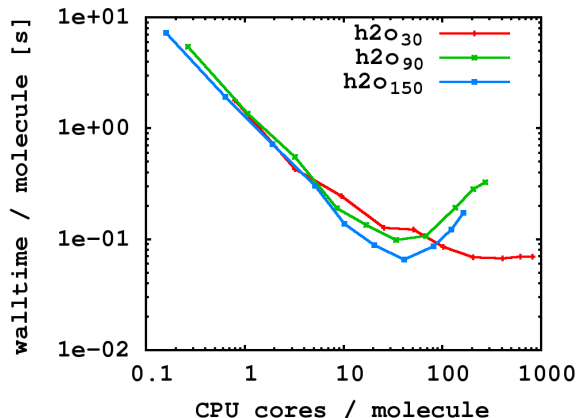
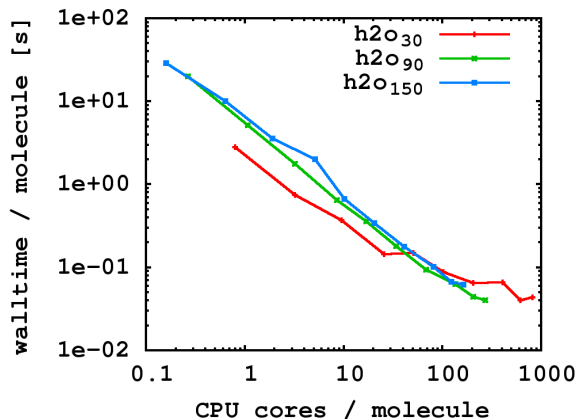


FIG. 7: The wall time of the last iteration of the SP2 algorithm at scale at $\tau = 10^{-8}$.



Typically, parallel performance of current matrix solvers goes out to a few molecules per core. SpAMM allows for a fine grained over-decomposition of the 3D product space, leading to a speedup far beyond this value. We observe that even at > 100 cores / molecule, i.e. a more than 100 fold increase in parallelism, the walltime per multiply decreases with increasing core count. This is only possible because of the over-decomposition in the 3-dimensional product space, as opposed to approaches that rely on the 2-dimensional vector space of the matrices.

V. CONCLUSIONS

In this contribution, we have explored two software approaches to parallel SpAMM within a quantum chemical solver. We find that both approaches demonstrate high parallel scalability and performance on a large SMP system and LANL's mustang computer cluster. The first approach is based on OpenMP and is therefore restricted to

FIG. 8: The wall time of the first iteration of the SP2 algorithm at scale at $\tau = 10^{-6}$.

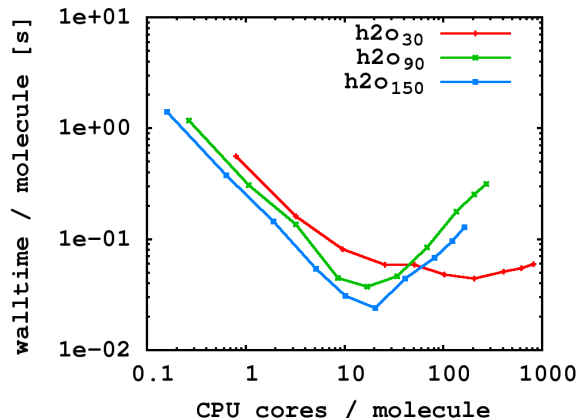
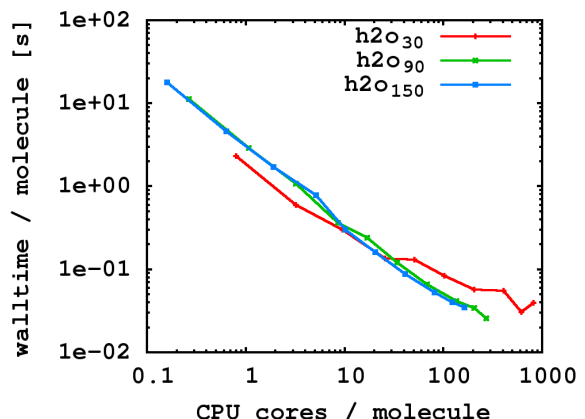


FIG. 9: The wall time of the last iteration of the SP2 algorithm at scale at $\tau = 10^{-6}$.



shared memory systems. Using an appropriate runtime implementation, we find that parallel scaling is close to ideal up to 24 cores and only slightly below ideal up to 48 cores. We speculate that the deviation from ideal as the number of threads is increased is mainly due to memory bandwidth bottlenecks and that SpAMM transitions from a compute bound implementation to being memory bound. The second approach uses the Charm++ framework and can be deployed on all major currently installed Top500 supercomputers. We find that the clean recursive implementation of the OpenMP approach can not be mapped directly onto Charm++ due to the lack of transient and fully load balanced task-like chares. Our Charm++ implementation therefore makes use of regular multi-dimensional chare arrays. Measuring the performance of the SP2 method on up to 24,000 cores of mustang, we find very good parallel scaling. We note that for the smaller water systems an allocation of over 24,000 cores translates into almost 1000 cores per water molecule. Finally, a comparison between the first and last iterations of SP2 demonstrates the performance of

the Charm++ load balancing algorithms, which dynamically and persistently migrate data and work throughout the computer cluster and significantly improve the performance of SpAMM throughout the SP2.

VI. ACKNOWLEDGEMENTS

We thank the LDRD program for funding this research under LDRD-ER grant 20110230ER. We would also like

to acknowledge generous support from the Ten-Bar Café providing stimulating and helpful discussions. This article was released under LA-UR-14-22050.

-
- [1] M. Challacombe and N. Bock, arXiv:cs.DS **1011.3534** (2010), 1011.3534.
 - [2] N. Bock and M. Challacombe, SIAM Journal on Scientific Computing **35**, C72 (2013), ISSN 1064-8275.
 - [3] P. Mishra and M. H. Eich, ACM Comput. Surv. **24**, 63 (1992).
 - [4] D. A. Schneider and D. J. DeWitt, in *Proceedings of the sixteenth international conference on Very large databases* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990), pp. 469–480.
 - [5] S. Chen, A. Ailamaki, P. B. Gibbons, and T. C. Mowry, ACM Trans. Database Syst. **32** (2007).
 - [6] C. Kim, T. Kaldewey, V. W. Lee, E. Sedlar, A. D. Nguyen, N. Satish, J. Chhugani, A. Di Blas, and P. Dubey, Proc. VLDB Endow. **2**, 1378 (2009).
 - [7] M. S. Warren and J. K. Salmon, SC Conference pp. 570–576 (1992).
 - [8] M. S. Warren and J. K. Salmon, in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing* (ACM, New York, NY, USA, 1993), Supercomputing '93, pp. 12–21.
 - [9] M. S. Warren and J. K. Salmon, Computer Physics Communications **87**, 266 (1995).
 - [10] M. Challacombe, E. Schwegler, and J. Almlöf, J. Chem. Phys. **104**, 4685 (1996).
 - [11] E. Schwegler, M. Challacombe, and M. Head-Gordon, The Journal of Chemical Physics **109**, 8764 (1998).
 - [12] L. Greengard and J. Strain, SIAM Journal on Scientific and Statistical Computing **12**, 79 (1991).
 - [13] J. Strain, SIAM Journal on Scientific and Statistical Computing **12**, 1131 (1991).
 - [14] B. J. C. Baxter and G. Roussos, SIAM Journal on Scientific Computing **24**, 257 (2002).
 - [15] C. Yang, R. Duraiswami, and N. Gumerov, Tech. Rep. CS-TR-4495, Dept. of Computer Science, University of Maryland, College Park (2003).
 - [16] X. Wan and G. E. Karniadakis, Journal of Computational Physics **219**, 7 (2006), ISSN 0021-9991.
 - [17] M. J. Berger and J. Oliger, Journal of Computational Physics **53**, 484 (1984), ISSN 0021-9991.
 - [18] J. Bell, M. J. Berger, J. Saltzman, and M. Welcome, SIAM Journal on Scientific Computing **15**, 127 (1994).
 - [19] M. J. Berger and P. Colella, Journal of Computational Physics **82**, 64 (1989), ISSN 0021-9991.
 - [20] L. Greengard and V. Rokhlin, Journal of Computational Physics **73**, 325 (1987), ISSN 0021-9991.
 - [21] L. Greengard and V. Rokhlin, Acta Numerica pp. 229–269 (1997).
 - [22] H. Cheng, L. Greengard, and V. Rokhlin, Journal of Computational Physics **155**, 468 (1999), ISSN 0021-9991.
 - [23] L. Greengard and V. Rokhlin, in *Vortex Methods*, edited by C. Anderson and C. Greengard (Springer Berlin / Heidelberg, 1988), vol. 1360 of *Lecture Notes in Mathematics*, pp. 121–141, ISBN 978-3-540-50526-6, 10.1007/BFb0089775.
 - [24] M. G. Choi, E. Ju, J.-W. Chang, J. Lee, and Y. J. Kim, Computer Graphics Forum **28**, 1773 (2009).
 - [25] T. Lewiner, V. Mello, A. Peixoto, S. Pesco, and H. Lopes, Computer Graphics Forum **29**, 1661 (2010).
 - [26] S. Lefebvre and H. Hoppe, ACM Trans. Graph. **25**, 579 (2006).
 - [27] S. Lefebvre and H. Hoppe, in *ACM SIGGRAPH 2006 Papers* (ACM, New York, NY, USA, 2006), SIGGRAPH '06, p. 579.
 - [28] Q. Avril, V. Gouranton, and B. Arnaldi, in *VRIC'09 Proceedings*, edited by S. R. . A. Shirai (Laval, France, 2009), vol. 11, p. 53.
 - [29] Y.-S. Zou, G.-F. Ding, M.-H. Xu, and Y. He, Application Research of Computers **25**, 8 (2008).
 - [30] J. Bittner, V. Havran, and P. Slavik, in *Computer Graphics International, 1998. Proceedings* (1998), pp. 207–219.
 - [31] Charm++, <http://charm.cs.uiuc.edu/>.
 - [32] A. Buluç and J. R. Gilbert, Arxiv preprint arXiv:1109.3739 (2011).
 - [33] A. Buluç and J. R. Gilbert, in *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing* (IEEE Computer Society, Washington, DC, USA, 2008), pp. 503–510.
 - [34] A. Buluç and J. R. Gilbert, in *2008 IEEE International Symposium on Parallel and Distributed Processing* (IEEE, 2008), pp. 1–11, ISSN 1530-2075.
 - [35] H. Menon and L. Kalé, in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis* (ACM, New York, NY, USA, 2013), SC '13, pp. 15:1–15:11, ISBN 978-1-4503-2378-9.
 - [36] O. Sarood, E. Meneses, and L. V. Kale, in *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA, 2013).
 - [37] L. V. Kale and A. Bhatele, eds., *Parallel Science and Engineering Applications: The Charm++ Approach* (Taylor & Francis Group, CRC Press, 2013), ISBN 9781466504127.
 - [38] J. Lifflander, S. Krishnamoorthy, and L. Kale, in *Proceed-*

- ings of the 34rd ACM SIGPLAN conference on Programming Language Design and Implementation, (To Appear)* (ACM, 2013), PLDI '13.
- [39] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. R. Quinn, in *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008* (2008).
 - [40] F. Gioachin, P. Jetley, C. L. Mendes, L. V. Kale, and T. R. Quinn, Tech. Rep. 07-08 (2007).
 - [41] C. Mei, Y. Sun, G. Zheng, E. J. Bohm, L. V. Kalé, J. C. Phillips, and C. Harrison, in *Proceedings of the 2011 ACM/IEEE conference on Supercomputing* (Seattle, WA, 2011).
 - [42] L. V. Kale, A. Bhatele, E. J. Bohm, and J. C. Phillips, in *Encyclopedia of Parallel Computing (to appear)*, edited by D. Padua (Springer Verlag, 2011).
 - [43] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale, Tech. Rep. UIUCDCS-R-2009-3034, Department of Computer Science, University of Illinois at Urbana-Champaign (2009).
 - [44] A. Szabo and N. Ostlund, *Modern quantum chemistry: introduction to advanced electronic structure theory* (Dover Pubns, 1996).
 - [45] R. McWeeny, P. Roy. Soc. Lond A Mat. **235**, 496 (1956).
 - [46] A. H. R. Palser and D. E. Manolopoulos, Phys. Rev. B **58**, 12704 (1998).
 - [47] M. Challacombe, J. Chem. Phys. **110**, 2332 (1999).
 - [48] M. Challacombe, Comp. Phys. Comm. **128**, 93 (2000).
 - [49] M. Benzi and N. Razouk, Electron. T. Numer. Ana. **28**, 16 (2007).
 - [50] M. Benzi, P. Boito, and N. Razouk, arXiv:math.NA **1203.3953** (2012).
 - [51] N. Bock, M. Challacombe, C. K. Gan, G. Henkelman, K. Nemeth, A. M. N. Niklasson, A. Odell, E. Schwegler, C. J. Tymczak, and V. Weber, *FREEON: A suite of programs for linear scaling quantum chemistry* (2011), <http://www.freeon.org/>.
 - [52] M. Challacombe, E. Schwegler, and J. Almlöf, in *Computational Chemistry: Reviews of Current Trends*, edited by J. Leszczynski (World Scientific, Singapore, 1996), vol. 1 of *Computational Chemistry: Reviews of Current Trends*, pp. 53–107.
 - [53] M. Challacombe and E. Schwegler, J. Chem. Phys. **106**, 5526 (1997).
 - [54] J. C. Burant, G. E. Scuseria, and M. J. Frisch, The Journal of Chemical Physics **105**, 8969 (1996).
 - [55] E. Schwegler, M. Challacombe, and M. Head-Gordon, J. Chem. Phys. **106**, 9708 (1997).
 - [56] J. M. Millam and G. E. Scuseria, The Journal of Chemical Physics **106**, 5569 (1997).
 - [57] A. D. Daniels, J. M. Millam, and G. E. Scuseria, The Journal of Chemical Physics **107**, 425 (1997).
 - [58] C. Ochsenfeld, C. A. White, and M. Head-Gordon, The Journal of Chemical Physics **109**, 1663 (1998).
 - [59] A. M. N. Niklasson, Phys. Rev. B **66**, 5 (2002).